# Active machine learning for spatio-temporal predictions using feature embedding

Arsam Aryandoust<sup>1\*</sup>, Stefan Pfenninger<sup>1</sup> December 9, 2020

1. Climate Policy Research Group, Department of Environmental Systems Science, Swiss Federal Institute of Technology (ETH Zurich) \*corresponding author: Arsam Aryandoust (arsam.aryandoust@usys.ethz.ch), CHN J 72.2, Universitaetstrasse 22, 8006 Zurich, Switzerland

#### Abstract

Active learning (AL) could contribute to solving critical environmental problems through improved spatio-temporal predictions. Yet such predictions involve high-dimensional feature spaces with mixed data types and missing data, which existing methods have difficulties dealing with. Here, we propose a novel batch AL method that fills this gap. We encode and cluster features of candidate data points, and query the best data based on the distance of embedded features to their cluster centers. We introduce a new metric of informativeness that we call *embedding entropy* and a general class of neural networks that we call *embedding networks* for using it. Empirical tests on forecasting electricity demand show a simultaneous reduction in average prediction RMSE by up to 63-88% and data usage by up to 50-69% compared to passive learning (PL) benchmarks.

Solving many of the world's most critical problems, especially environmental ones, involves the accurate prediction of some value of interest [1]. Examples include the electricity consumption of buildings, required to operate sustainable power grids; the travel time between city zones, required for the smart charging of electric vehicles; and meteorological conditions, required for weather-based forecasting of wind and solar electricity generation. Sensing and labeling the ground truth data that is necessary for making these predictions in time and space usually comes at a high cost. This cost constrains the total number of sensors that we can place and use to query new data. A fundamental question that arises for many spatio-temporal prediction tasks is where and when to measure and query the data required to make the best possible predictions while staying within a maximum budget for sensors and data.

Active learning (AL) provides solutions to this question. It deals with the task of making accurate predictions using limited data when measuring and labeling data is expensive, and the amount of candidate data points for training a prediction model is very large and/or unequally distributed. In contrast, passive learning (PL) represents the widely used method in which candidate data points for training a prediction model are chosen at random. Existing AL methods have the goal to query the most informative unlabeled data point(s), and can roughly be categorized into two groups [2, 3, 4]: a first group of methods tries to shrink the space of candidate prediction models as fast as possible, while a second group of methods tries to exploit patterns in the underlying data. The first group of methods can further be categorized into uncertainty [5, 6, 7, 8, 9, 10, 11, 12], disagreement [13, 14, 15, 16, 17, 18, 19], variance reduction [20, 21, 22, 23, 24, 25], error reduction [26, 27, 28, 29] and model change based [30, 31] methods. The second group of methods is less studied and distinguishes mainly density weighted [30, 31] and clustering based [32, 33, 34] methods.

Each of these existing AL methods has its own strengths and limitations, but none of them are applicable to spatio-temporal prediction problems in which (i) uncertainty measures are missing, such as in most regression models, (ii) the underlying data are heterogeneous and high-dimensional, such as data consisting of both time-series and images, (iii) the parameter space of candidate prediction models is large, such as with deep (recurrent/convolutional) neural networks, and (iv) not only labels but also features can be missing, and must be queried at a potentially high cost, such as with high resolution aerial imagery. While all methods referred to above become infeasible under (iv), each of the remaining properties prevents one set of methods from being feasible: uncertainty based methods require explicit measures of uncertainty about a prediction which is not given under (i); disagreement based methods need to simultaneously train several models, which becomes expensive under (ii); variance reduction based methods must calculate the inverse of a K X K matrix where K is the number of model parameters, which becomes impractical under (iii); model change and error reduction based methods need to evaluate model weights and predictions for each candidate data point, which becomes expensive under (ii) and (iii). The second group of methods becomes mainly impractical under (ii) if there exists mutual information between features of different data types that cannot be compared in the raw feature space.

Here, we propose a novel AL approach in which the distances between the features of candidate data points in an embedded vector space are used to create priors on the true labels of candidates for clustering and querying the most informative data. We call this metric the *embedding entropy* of candidate data points and introduce a class of neural networks that we call *embedding networks* for using it. The algorithm that we propose is motivated by the observation that vector distances of embedded features preserve an increasing amount of information about the context of the data by which they are trained [35, 36, 37, 38, 39, 40]. It combines the strengths of both worlds of currently existing AL methods, making effective data queries feasible for a wider range of prediction tasks including those outlined in (i) - (iv). Figure 1 provides an overview of our method, including the general architecture of an embedding network (a.) and different variants of the embedding entropy that we use for querying candidate data points (b.-d.).

We test the performance of our AL algorithm against PL scenarios when predicting the electric load profile of a building  $\mathbf{y}_{t,s}$  given an aerial image of that building  $\mathbf{x}_s$ , the meteorological conditions at that building  $\mathbf{x}_{st}$  and a time stamp  $\mathbf{x}_t$ , which together shape our joint feature vector  $\mathbf{x}_{t,s}$ . This is a real-world problem to which the characteristics of (i)-(iv) apply. We start with a prediction model that has learnt this relationship for a few buildings and times. In each iteration of our AL algorithm, we query a batch of new data points from a candidate data pool for training a neural network prediction model. We assume our data budget to be 50% of the size of our candidate data pool, and perform 10 iterations of AL where we query 10% of our data budget in each iteration. We distinguish between three prediction types: spatial (load in a building without a sensor but where we have simultaneous data from other buildings), temporal (load in a building that has a sensor but for a future time period) and spatiotemporal (load for times and buildings where we have no data available at all). For each prediction type, we distinguish between the feature vectors  $\mathbf{x}_t$ ,  $\mathbf{x}_s$ ,  $\mathbf{x}_{st}$ and  $\mathbf{x}_{t,s}$  that we can encode for querying candidate data points. We further test AL for querying candidates based on the embedding entropy of their true labels  $\mathbf{y}_{t,s}$ , representing the optimal priors that we can build using our features and the optimal queries that we can make using our metric. We refer to these as our AL variables. In each experiment, we test our algorithm for randomizing (Figure 1, b.), maximizing (Figure 1, c.) and minimizing (Figure 1, d.) the embedding entropy of candidate data points in the queried data batch. We refer to these as our AL variants.

## Numeric results

We can reduce our data demand by up to 50-69% while making up to 63-88% more accurate predictions compared to PL benchmarks. When we exploit our entire data budget, we make up to 82-86% more accurate predictions compared to the PL benchmarks, and up to 73-79% more accurate predictions compared

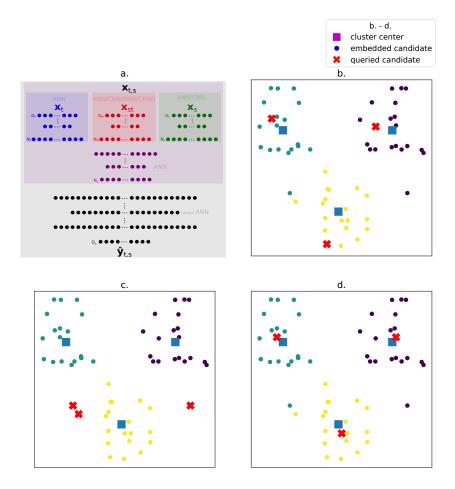


Figure 1: An overview of our AL method. a.) A general embedding network architecture where the abbreviation ANN stands for a densely connected, CNN for a convolutional, RNN for a recurrent and CRNN for a convolutional and recurrent neural network architecture. b.) - d.) represent the candidate data points that are encoded using one of the encoders from the embedding network for a queried batch size of three and embedded feature vectors of dimension two. Points of the same color are equally clustered. Squares represent cluster centers. Crosses describe which data points are chosen when b.) randomizing, c.) maximizing and d.) minimizing the embedding entropy of a queried data batch in an iteration of our algorithm.

to when saving data. We find that a reduced sensor demand of 2-8% is possible with priors on the true labels of candidate data points but is not achieved by our algorithm in practice when using embedded features for querying data points. The improved prediction accuracy that we report refers to the average root mean squared error (RMSE) between predicted and true electric load profiles of data points that remain unqueried after our AL algorithm terminates. We refer to these as our test losses. Figure 1 visualizes the performance of our initial model, a passively extended model and an actively extended model on exemplar data points. While both passively and actively trained models make significant improvements compared to our initial model, we can observe that we catch trends better with AL than with PL.

Table 1 contains the numeric results of the experiments that we conduct for each prediction type, AL variable and AL variant. We remove queried data points from the candidate data pool at a rate  $\delta$ . Keeping queried points, i.e. allowing the algorithm to re-use them, lets us implicitly remove any initial or ongoing bias from our model and further reduce data usage. We can observe that when removing all queried data points from the candidate data pool ( $\delta = 1$ ), our algorithm primarily reduces the testing loss. When keeping all queried data points in the candidate data pool ( $\delta = 0$ ), it mainly reduces data usage. The decrease in both data usage and testing loss is similar across all prediction types. The only exception is for temporal predictions and the AL variables  $\mathbf{x}_{t,s}$  and  $\mathbf{y}_{t,s}$ , where the decrease in testing loss is significantly larger compared to their PL benchmark, mainly because of a weak performance of PL. Hence we neglect these results in the improvements that we report.

We observe that the AL variable  $\mathbf{x}_{t,s}$  generally gets closest to these optimal results. When using  $\mathbf{x}_{st}$  as our AL variable, we achieve no or only a small decrease of the testing loss with the same data usage ( $\delta = 1$ ), or a significant decrease of the data usage ( $\delta = 0$ ) at a strongly decreased (spatial), moderately decreased (spatio-temporal) and partly increased (temporal) testing loss. With the AL variables  $\mathbf{x}_t$  and  $\mathbf{x}_s$ , we generally make equally good or slightly worse predictions than the PL benchmark. For  $\delta = 0$ ,  $\mathbf{x}_t$  and  $\mathbf{x}_s$  either use the same amount of data as the PL benchmark (random  $H^{NN}$ ), or an amount of data that is close to the batch size of data points that we query in each iteration of the algorithm (max/min  $H^{NN}$ ). For  $\delta = 1$ , all AL variants for  $\mathbf{x}_t$  and  $\mathbf{x}_s$ perform similar to PL. This is because the diversities of  $\mathbf{x}_t$  and  $\mathbf{x}_s$  are smaller than the batch size and number of clusters in the data points that we query in each iteration of our AL algorithm. Therefore our algorithm performs identical to the PL benchmarks for these variables with  $\delta = 1$ . For  $\delta = 0$ , when we collect candidate data points evenly distributed from all clusters, we also collect data points uniformly at random from the entire candidate data pool and hence proceed equal to PL. If instead we maximize or minimize the embedding entropy of candidate data points, we choose about the same set of data points in each iteration and hence observe a data usage that is close to the batch size of the data that we query in a single iteration (10-15%).

The largest decrease in testing loss with prior knowledge on true labels  $\mathbf{y}_{t,s}$  is generally achieved for the AL variant of maximizing embedding entropy (max

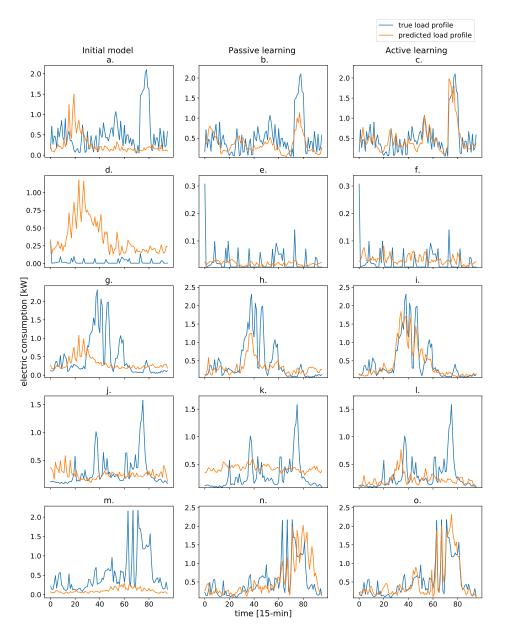


Figure 2: Exemplar spatio-temporal predictions prior to extending model on candidate data (Initial model), once after passively extending model (Passive learning), and once after actively extending model with randomized embedding entropy (max  $H^{NN}$ ) of jointly encoded features  $x_{t,s}$  (Active learning). Each row contains the same true load profile (blue line) but different predictions of it (orange line).

 $H^{NN}$ ). The largest practical decrease in testing loss is achieved for  $\mathbf{x}_{t,s}$  and the AL variant of choosing embedding entropy evenly at random (random  $H^{NN}$ ). The largest decrease in data usage, without increasing testing loss compared to our PL benchmark, is generally achieved for the AL variants of maximizing (max  $H^{NN}$ ) and minimizing (min  $H^{NN}$ ) embedding entropy. In all cases, the AL variant of choosing the embedding entropy of the candidate data batch uniformly at random (random  $H^{NN}$ ) achieves a lower prediction loss at the price of a higher data usage compared to the other AL variants. Among the AL variants of the same AL variable, we can observe a trade-off between higher data usage and lower testing losses. In the following, we focus on the results for the AL variables  $\mathbf{x}_{st}$ ,  $\mathbf{x}_{t,s}$  and  $\mathbf{y}_{t,s}$ , and the most difficult of all three prediction types: spatio-temporal.

# Training & validation losses for removing queried data ( $\delta = 1$ )

For validations on unqueried data points during training, Figure 2 shows the training and validation losses of our AL variables and variants for spatio-temporal predictions in which we remove queried data points from the candidate data set  $(\delta=1)$  compared to our PL benchmark (dashed blue line). All AL variants of the same AL variables behave identical. In the case of querying candidate data based on labels  $\mathbf{y}_{t,s}$ , we observe the largest training losses in the first iterations of the algorithm compared to the other AL variables. Training losses decline towards their PL benchmark values, while validation losses are constantly lower than the PL benchmark; they build an increasing gap to PL losses with each iteration of the algorithm. Validation losses drop step-wise at the beginning of each iteration of the algorithm and converge to their respective test loss.

Training and validation losses of the AL variables  $\mathbf{x}_{st}$  and  $\mathbf{x}_{t,s}$  are similar in the first iteration of the algorithm. Training losses are higher than the PL benchmark and decrease at the same rate as for the PL benchmark. Validation losses are slightly smaller than for PL and also decrease at the same rate as for PL. In the following iterations, training and validation losses continue to decrease for  $\mathbf{x}_{st}$  until the training losses become smaller than for PL and the gap of validation losses to the PL benchmark increases. For  $\mathbf{x}_{t,s}$ , we observe a leap in training loss and a simultaneous drop of validation loss at the beginning of each iteration. These leaps and drops of training and validation loss become smaller in magnitude with each iteration of the algorithm. Hence we implicitly regularize our model weights as our AL model does not overfit to the queried data points and makes overall better predictions on unqueried points than our PL model.

# Training & validation losses for keeping queried data ( $\delta = 0$ )

For validations on unqueried data points during training, Figure 3 shows the training and validation losses for keeping queried data points in the candidate data pool ( $\delta = 0$ ) compared to our PL benchmark (dashed blue line). Unlike in the case of removing candidates ( $\delta = 1$ ), the training loss of querying data

Table 1: The numeric results of our AL experiments for each prediction type, AL variable and AL variant. AL variables describe which (partial) feature vector is encoded, clustered and used to calculate the embedding entropy  $H^{NN}$  of candidate data points. AL variants describes whether the embedding entropy of candidate data points in each queried batch is randomized, maximized or minimized. The table states results for removing ( $\delta=1$ ) and keeping ( $\delta=0$ ) queried data in the candidate data pool. The columns named 'data' present how much of the data budget is used, 'sensors' state how many new sensors are used compared to how many new sensors are initially available in the candidate data pool, and 'test loss' shows the average RMSE between predicted and true electric load profiles for all candidate data points whose labels have not been queried when our AL algorithm terminates.

			Removing queried data $(\delta = 1)$			Keeping queried data ( $\delta = 0$ )		
prediction type	AL variable	AL variant	data	sensors	test loss [kW]	data	sensors	test loss [kW]
Spatial	$\mathbf{x}_t$	random $H^{NN}$	100%	100%	0.1952	80%	100%	0.7165
		$\max H^{NN}$	100%	100%	0.218	12%	100%	0.9644
		$\min H^{NN}$	100%	100%	0.2208	12%	100%	0.8948
	$\mathbf{x}_{s}$	random ${\cal H}^{NN}$	100%	100%	0.2182	80%	100%	0.8505
		$\max H^{NN}$	100%	100%	0.2288	10%	100%	1.1369
		$\minH^{NN}$	100%	100%	0.2125	10%	100%	0.9745
	$\mathbf{x}_{st}$	random $H^{NN}$	100%	100%	0.2002	47%	100%	0.2488
		$\max H^{NN}$	100%	100%	0.2117	13%	100%	0.4248
		$\min H^{NN}$	100%	100%	0.2159	13%	100%	0.4296
	$\mathbf{x}_{t,s}$	random $H^{NN}$	100%	100%	0.0373	49%	100%	0.1
		$\max H^{NN}$	100%	100%	0.0388	30%	100%	0.142
		$\min H^{NN}$	100%	100%	0.0384	30%	100%	0.1338
	$\mathbf{y}_{t,s}$	random $H^{NN}$	100%	100%	0.0309	43%	100%	0.2979
		$\max H^{NN}$	100%	100%	0.0305	18%	94%	0.3784
		$\min H^{NN}$	100%	100%	0.0353	17%	92%	0.4687
passive learning (PL) benchmark			100%	100%	0.2129	80%	100%	0.8477
Temporal	$\mathbf{x}_t$	random $H^{NN}$	100%	0%	1.7072	80%	0%	0.579
		$\max H^{NN}$	100%	0%	1.5321	14%	0%	0.7389
		$\minH^{NN}$	100%	0%	1.6294	15%	0%	0.7006
	$\mathbf{x}_s$	random $H^{NN}$	100%	0%	1.6734	81%	0%	0.5507
		$\max H^{NN}$	100%	0%	1.6998	10%	0%	0.7793
		$\min H^{NN}$	100%	0%	1.6075	10%	0%	0.7881
	$\mathbf{x}_{st}$	random $H^{NN}$	100%	0%	1.2799	75%	0%	0.4434
		$\max H^{NN}$	100%	0%	1.3421	13%	0%	0.6447
		$\minH^{NN}$	100%	0%	1.176	12%	0%	0.6408
	$\mathbf{x}_{t,s}$	random $H^{NN}$	100%	0%	0.0692	51%	0%	0.2109
		$\max H^{NN}$	100%	0%	0.0721	30%	0%	0.3441
		$\min H^{NN}$	100%	0%	0.0733	30%	0%	0.3409
	$\mathbf{y}_{t,s}$	random ${\cal H}^{NN}$	100%	0%	0.0624	43%	0%	0.2918
		$\max H^{NN}$	100%	0%	0.0612	19%	0%	0.5356
		$\min H^{NN}$	100%	0%	0.0643	20%	0%	0.5
passive learning (PL) benchmark			100%	0%	1.9526	80%	0%	0.5713
Spatio-temporal	$\mathbf{x}_t$	random $H^{NN}$	100%	100%	0.4573	80%	100%	0.9756
		$\max H^{NN}$	100%	100%	0.4428	12%	100%	1.0156
		$\minH^{NN}$	100%	100%	0.4576	12%	100%	0.9974
	$\mathbf{x}_{s}$	random $H^{NN}$	100%	100%	0.4292	80%	100%	0.9034
		$\max H^{NN}$	100%	100%	0.4647	10%	100%	1.0077
		$\min H^{NN}$	100%	100%	0.4412	10%	100%	1.022
	$\mathbf{x}_{st}$	random ${\cal H}^{NN}$	100%	100%	0.3533	57%	100%	0.7125
	**	$\max H^{NN}$	100%	100%	0.3477	12%	100%	0.8577
		$\min H^{NN}$	100%	100%	0.3463	12%	100%	0.8964
	$\mathbf{x}_{t,s}$	random $H^{NN}$	100%	100%	0.0593	45%	100%	0.1912
	-3	$\max H^{NN}$	100%	100%	0.0614	30%	100%	0.2679
		$\min H^{NN}$	100%	100%	0.0641	31%	100%	0.2628
	$\mathbf{y}_{t,s}$	random $H^{NN}$	100%	100%	0.053	37%	98%	0.4323
	<b>•</b> •,•	$\max H^{NN}$	100%	100%	0.0506	17%	97%	0.5927
		$\min H^{NN}$	100%	100%	0.0603	17%	97%	0.5927
passive learning (PL) benchmark			100%	100%	0.4442	81%	100%	0.9340
1								

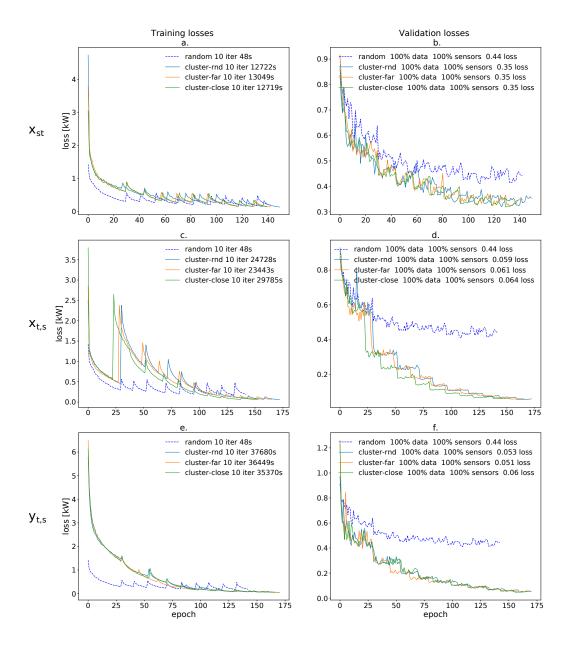


Figure 3: Training and validation losses of the AL variables  $\mathbf{x}_{st}$ ,  $\mathbf{x}_{t,s}$  and  $\mathbf{y}_{t,s}$ , and all AL variants compared to the PL benchmark (dashed blue lines) where every queried data point is removed from the candidate data pool for future AL iterations ( $\delta = 1$ ) exemplary shown for spatio-temporal predictions. The line legends for training losses show how many iterations of AL are performed and how much time they need to execute. The line legends for validation losses show which share of the data and sensor budget is used, and how large the testing losses are. Training stops at different epochs due to stochastic effects of early stopping (patience 10).

points based on the embedding entropy of their true labels  $\mathbf{y}_{t,s}$  converges to a value that is higher than the PL benchmark. Validation losses stay about constant for the AL variants of maximized (cluster-far) and minimized (cluster-close) embedding entropy and only decline for randomized embedding entropy (cluster-rnd) after the first iteration of the algorithm. The initial gaps between the validation loss of the PL benchmark and all AL variants, however, is larger than when removing candidates ( $\delta = 1$ ).

Similar to our experimental tests with prior knowledge on the true labels  $\mathbf{y}_{t,s}$ , the validation losses for  $\mathbf{x}_{t,s}$  and  $\mathbf{x}_{st}$  only decline for randomized embedding entropy (cluster-rnd) after the initial iterations of the algorithm. Unlike in the case of removing queried data ( $\delta = 1$ ), the training losses for  $\mathbf{x}_{t,s}$  converge to a value that is larger than for the PL benchmark, and validation losses for  $\mathbf{x}_{st}$  converge to similar values as for the PL benchmark when maximizing (cluster-far) and minimizing (cluster-close) embedding entropy.

## Validation losses against queried and unqueried candidate data

We now also consider the case of validating on both queried and unqueried data points during training. Figure 4 shows the validation losses for both removing  $(\delta = 1)$  and keeping  $(\delta = 0)$  queried data in the candidate pool for this case. For  $\delta = 1$ , we observe a different behaviour than when validating against the unqueried candidate data points only. Losses decrease along for  $\mathbf{x}_{st}$  or below the PL benchmark for  $\mathbf{x}_{t,s}$  and  $\mathbf{y}_{t,s}$  during the first iterations of the algorithm and increase slightly for  $\mathbf{x}_{st}$  or strongly for  $\mathbf{x}_{t,s}$  and  $\mathbf{y}_{t,s}$  above the PL benchmark values towards the last iterations of the algorithm. We create a bias towards the remaining candidate data points, which makes predictions on already queried data points less accurate.

When keeping queried candidates ( $\delta = 0$ ), validation losses for  $\mathbf{x}_{st}$  decrease equal to the PL benchmark for randomized embedding entropy (cluster-rnd) and stay about constant after the first iteration of the algorithm for maximized (cluster-far) and minimized (cluster-close) embedding entropy. This is different to when validating against the unqueried data only, as validation losses for maximized (cluster-far) and minimized (cluster-close) embedding entropy are not equal to the PL benchmark anymore but higher, and losses for randomized embedding entropy (cluster-rnd) do not decrease below the PL benchmark but are identical to it. For  $\mathbf{x}_{t,s}$  and  $\mathbf{y}_{t,s}$ , validation losses behave identical to when validating against unqueried candidates. Hence we correct the bias of our model towards remaining candidate data points and improve predictions on unqueried data points compared to when removing queried candidates ( $\delta = 1$ ).

#### Discussion

We investigate whether embedded feature vectors can provide a useful metric of informativeness for reducing the data demand in spatio-temporal prediction tasks in which (i) uncertainty measures are missing, (ii) the underlying data is mixed and high dimensional, (iii) the parameter space of candidate prediction

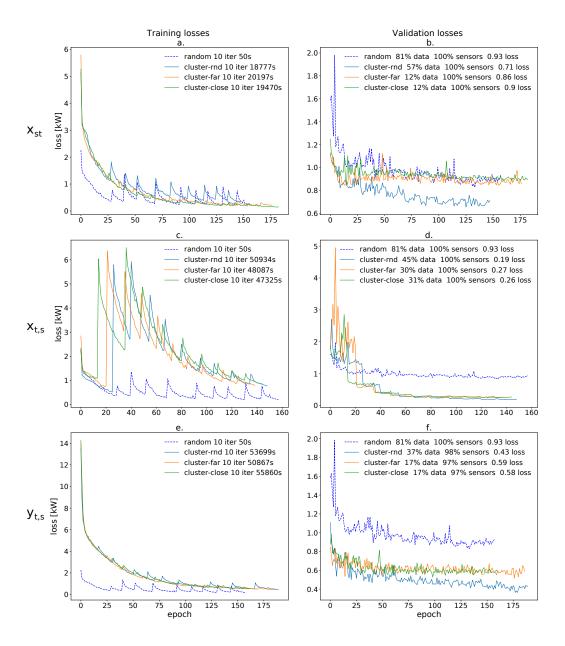


Figure 4: Training and validation losses of the AL variables  $\mathbf{x}_{st}$ ,  $\mathbf{x}_{t,s}$  and  $\mathbf{y}_{t,s}$ , and all AL variants compared to the PL benchmark (dashed blue lines) where every queried data point is removed from the candidate data pool for future AL iterations ( $\delta = 0$ ) exemplary shown for spatio-temporal predictions. The line legends for training losses show how many iterations of AL are performed and how much time they need to execute. The line legends for validation losses show which share of the data and sensor budget is used, and how large the testing losses are. Training stops at different epochs due to stochastic effects of early stopping (patience 10).

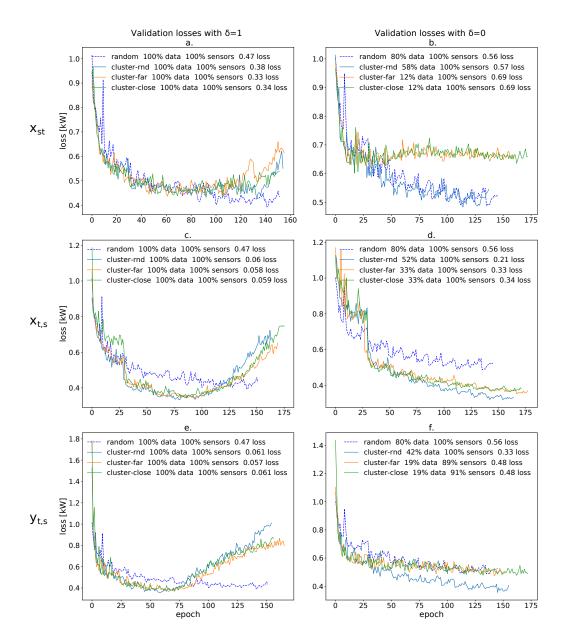


Figure 5: Validation losses against both queried and unqueried candidate data points of the AL variables  $\mathbf{x}_{st}$ ,  $\mathbf{x}_{t,s}$  and  $\mathbf{y}_{t,s}$ , and all AL variants compared to their PL benchmarks (dashed blue lines) when removing ( $\delta = 1$ ) and keeping ( $\delta = 0$ ) queried data points in the candidate data pool for spatio-temporal predictions. The line legends show which share of the data and sensor budget is used, and how large the testing losses are. Training stops at different epochs due to stochastic effects of early stopping (patience 10).

models is large, and (iv) not only labels but also features are missing or must be queried at a high cost. We find that clusters in embedded feature vectors can provide such a metric, and that the distance between encoded feature vectors and their cluster centers can be used to query informative data from a large pool of candidate points. AL has not been studied for the settings in (i) - (iv), so our findings are the first answer to this research question. Our results suggest that AL could contribute to solving spatio-temporal prediction tasks that will help tackle climate change and other urgent environmental problems: electricity demand, as in our example application, but also many other problems such as the generation of electricity under fluctuating renewable energy sources which is important for faster decarbonizing our grids. This encourages further research on how to use machine learning and artificial intelligence more generally in this and many of the related domains as it is also suggested by [1].

We observe that the prediction accuracy of our model can be increased using fewer data points compared to PL benchmarks. This is consistent with the findings of recent studies, where a higher classification accuracy is achieved with the same or a smaller amount of data points using similar AL methods such as density clustering and other variants of information entropy [41, 42, 43]. However, none of these studies report improvements in prediction accuracy that are as large as we find here. We further find that large improvements can be achieved using less sensors, and when querying data points based on the embedding entropy of single feature parts only. Hence we suspect our algorithm provides the first AL method that can deal with missing data in the feature space, and which suggests that not only the demand for data but also the demand for sensors can be reduced in spatio-temporal prediction tasks.

The metric that we propose has many similarities to existing AL methods. When we randomize the embedding entropy of our queried data batch (Figure 1, b.), our algorithm performs importance sampling, with the difference that we can decide on the importance of learned feature maps in the context of our prediction problem instead of our raw data [44]. When we maximize embedding entropy, our algorithm performs uncertainty sampling, with the difference that we do not need an explicit metric of uncertainty as it is the case for Gaussian processes for instance. We likely sample data points at our decision boundaries (Figure 1, c.), with the difference that unlike in classification problems, we do not need explicit decision boundaries [45]. When we minimize embedding entropy, our algorithm queries data points that are most representative for each cluster in our candidate data (Figure 1, d.), which is similar to performing core-set sampling [46]. When we query candidates based on the embedding entropy of predicted labels, this is similar to pursuing G-optimality in optimal experimental design [47, 48]. It implies that our metric uses elements of model change and error reduction based methods where a prediction model, which is optimized using gradient decent, is considered to be uncertain if knowing the label induces a large gradient of the loss and a large update to model parameters [49].

The prediction task and the data that we use for our tests are specific. To make general statements about our algorithm's reliability, the performance of our algorithm and our metric of entropy must be tested on other datasets

and prediction tasks. One study that has proposed a similar method finds consistently high performance across different datasets and architectures [49]. [49] find that their method outperforms or performs at least as well as existing AL methods where these are applicable. The method that they propose clusters and queries samples based on gradients of a deep learning network's last layer embedding of candidate data points. This is equal to the case in our method in which we randomize the embedding entropy of the entire feature vector, and our joint encoder contains all except the last layer of our embedding network (Figure 1, a.). The results from [49] thus encourage further research on applying our method to prediction types beyond the scope of spatio-temporal ones.

Further research should explore the existence of theoretical guarantees for the batch AL algorithm and embedding entropy that we propose. We show that sensor savings are possible with prior knowledge about the labels of candidate data points, but more work is needed on how these results can be practically achieved. Optimal strategies can be developed for removing queried data points from the candidate data pool so as to balance the trade-off between our desire to lower sensor and data demand on the one hand, and retaining an acceptable prediction accuracy on the other hand. Using an adaptive drop-out rate  $\delta$  may provide more control over correction of bias towards queried data. Finally, we may reduce the computational complexity of our algorithm through for instance transforming our AL problem to a stream-based one as suggested by [50].

# Methods

Given the aerial image of a building, the meteorological conditions in the region of that building and a time stamp, we want to predict the electric load profile of the building for the next 24 hours in 15 min steps. We start with a prediction model that has learnt this relationship for a few buildings and times. Our features are all remotely sensed and assumed to be available for every building and point in time at no cost. For every new load profile that we collect, we assume to experience some cost and are constrained in the total number of profiles that we can collect by some budget  $n_{budget}$ . Our goal is to collect further ground truth data, i.e. the electric load profiles at different times and buildings, so as to make the best possible predictions for buildings and times, for which we do not have load profiles available, without exceeding  $n_{budget}$ .

In each iteration of the AL algorithm that we propose, we query a batch of candidate data points. First, we encode the features of candidate data points into an embedded vector space using a modular neural network prediction model that is trained on initially available data points. We then cluster candidate data points based on their vector distances to each other in this encoded space. Next, we use the distance of the vector of each encoded data point to its cluster center to introduce a new metric of informativeness that we call the *embedding entropy*. Candidate data points whose embedded feature vectors are further away from their cluster's center are expected to be more diverse than other members and therefore have a larger embedding entropy. We expect to be more uncertain

about these points, as they are more likely to be true members of another cluster: we likely explore the data that is close to our *decision boundaries*, if not outliers, and expect a larger *surprise* from querying labels for these data points. When querying labels of data points that are close to their cluster centers, we expect these to be more *representative* of their clusters and all our entire candidate data, respectively. We test our method for randomized (random  $H^{NN}$ ), maximized (max  $H^{NN}$ ) and minimized (min  $H^{NN}$ ) embedding entropy, and refer to these as our AL variants.

We evaluate the performance of our algorithm for spatial, temporal and spatio-temporal predictions compared to a PL benchmark. In this context, temporal predictions mean that we predict the load profile for buildings in which a sensor is placed, but for a time period into the past or future, for which we do not have measured data available. Spatial predictions mean that we predict a load profile for buildings in which a sensor is not placed, but for a time period in which we do have load profiles available for other buildings. Spatio-temporal predictions respectively refer to the most difficult problem of predicting load profiles for times and buildings, for which we do not have any load profiles available at all. We refer to these three variants as prediction types.

For each prediction type that we evaluate, we further distinguish between the type of features that we can encode for querying candidate data points. We distinguish between features that are variant in time  $\mathbf{x}_t$  (time stamp), space  $\mathbf{x}_s$  (building image), and both time and space  $\mathbf{x}_{st}$  (meteorological data), as well as the entire feature vector  $\mathbf{x}_{t,s}$  which is concatenated from these three vectors. In a further test, we use the embedding entropy of our labels  $\mathbf{y}_{t,s}$  for querying candidate data points so as to see how our proposed metric performs with priors on the true labels of candidate data. We refer to these vectors as AL variables.

# The spatio-temporal prediction problem

Given a map of the Earth, we want to predict some value of interest  $\mathbf{y}_{t,s} \in \mathbb{R}^{D_y}$  of dimension  $D_y \in \mathbb{Z}^+$  in time  $t \in \mathbb{N}$  and space  $\mathbf{s} \in \mathbb{R}^2$  such that  $\mathbf{s} = (lat, long)$ , with  $lat \in [-90, 90]$  and  $long \in [-180, 180]$ . The ranges of the variables lat and long refer to the possible values of geographic latitudinal and longitudinal coordinates. Hereby, the starting point in time, and the accuracy in both time and space are application dependent and can be chosen arbitrarily. The set of all  $\mathbf{y}_{t,s}$ , hereafter called labels, is referred to as  $\mathcal{Y}$ . Each label is hence a vector

$$\mathbf{y}_{t,s} = \begin{pmatrix} y_{t,s,1} \\ \vdots \\ y_{t,s,D_y} \end{pmatrix} \tag{1}$$

Given the features  $\mathbf{x}_{t,s} \in \mathbb{R}^{D_x}$  of dimension  $D_x \in \mathbb{Z}^+$  for each label, we want to predict labels for particular points of interest in time and space. We refer to the set of all features as  $\mathcal{X}$ . Each label hence has a corresponding feature vector

$$\mathbf{x}_{t,s} = \begin{pmatrix} x_{t,s,1} \\ \vdots \\ x_{t,s,D_x} \end{pmatrix} \tag{2}$$

We can further classify the single entries of  $\mathbf{x}_{t,s}$  as space, time and spacetime variant features. Features that are constant in time t but variant in space  $\mathbf{s}$  are referred to as space variant features  $\mathbf{x}_s \in \mathbb{R}^{D_s}$  of dimension  $D_s \in \mathbb{N}$  such that  $D_s \leq D_x$ :

$$\mathbf{x}_{s} = \begin{pmatrix} x_{t,s,1} \\ \vdots \\ x_{t,s,D_{s}} \end{pmatrix} = \begin{pmatrix} x_{t+\hat{t},s,1} \\ \vdots \\ x_{t+\hat{t},s,D_{s}} \end{pmatrix} \forall \hat{t} \in \mathbb{Z} : 0 \le t + \hat{t}$$
 (3)

Features that are constant in space **s** but variant in time t are referred to as time variant features  $\mathbf{x}_t \in \mathbb{R}^{D_t}$  of dimension  $D_t \in \mathbb{N}$  such that  $D_t \leq D_x$ :

$$\mathbf{x}_{t} = \begin{pmatrix} x_{t,s,D_{s}+1} \\ \vdots \\ x_{t,s,D_{s}+D_{t}} \end{pmatrix} = \begin{pmatrix} x_{t,s+\hat{s},D_{s}+1} \\ \vdots \\ x_{t,s+\hat{s},D_{s}+D_{t}} \end{pmatrix}$$

$$\forall \hat{\mathbf{s}} \in \mathbb{R}^{2} : lat + l\hat{a}t \in [-90, 90] \land long + long \in [0, 180]$$

$$(4)$$

Features that are variant in both time t and space  $\mathbf{s}$  are referred to as space-time variant features  $\mathbf{x}_{st} \in \mathbb{R}^{D_{st}}$  of dimension  $D_{st} \in \mathbb{N}$  such that  $D_{st} \leq D_x$ :

$$\mathbf{x}_{st} = \begin{pmatrix} x_{t,s,D_s+D_t+1} \\ \vdots \\ x_{t,s,D_s+D_t+D_{st}} \end{pmatrix} \neq \begin{pmatrix} x_{t+\hat{t},s+\hat{s},D_s+D_t+1} \\ \vdots \\ x_{t+\hat{t},s+\hat{s},D_s+D_t+D_{st}} \end{pmatrix}$$
(5)

 $\forall \hat{t} \in \mathbb{Z}, \hat{\mathbf{s}} \in \mathbb{R}^2 : 0 \le t + \hat{t} \land lat + \hat{lat} \in [-90, 90] \land long + \hat{long} \in [0, 180]$ 

For the above to be valid, it further has to hold that

$$D_s + D_t + D_{st} = D_x \tag{6}$$

and

$$\mathbf{x}_{t,s} = \begin{pmatrix} \mathbf{x}_t \\ \mathbf{x}_s \\ \mathbf{x}_{st} \end{pmatrix} \tag{7}$$

We can further distinguish the type of predictions that we make in the same fashion. Let  $\mathcal{Y}^{avail} \subset \mathcal{Y}$  be a subset of all labels that are available to us, i.e. the ground truth values that we have already measured. Then, for a given point  $\mathbf{s}$  in space, if we use any knowledge about  $\mathbf{y}_{t_1,s}$  such that  $t_1$  is represented in elements of  $\mathcal{Y}^{avail}$  to make predictions about any  $\mathbf{y}_{t_2,s}$  such that  $t_2$  is not represented

by elements of  $\mathcal{Y}^{avail}$ , we call this a temporal prediction. Similarly, for a given point t in time, if we use any knowledge about  $\mathbf{y}_{t,s_1}$  such that  $\mathbf{s}_1$  is represented by elements of  $\mathcal{Y}^{avail}$  to make predictions about any  $\mathbf{y}_{t,s_2}$  such that  $\mathbf{s}_2$  is not represented by elements of  $\mathcal{Y}^{avail}$ , we call this a spatial prediction. In contrast, if we make predictions about any  $\mathbf{y}_{t,s}$  such that  $(t,\mathbf{s})$ , is not represented by elements of  $\mathcal{Y}^{avail}$ , we call this a spatio-temporal prediction.

Furthermore, let  $\mathcal{X}^{avail}$  be the set of features that are complement to each element of  $\mathcal{Y}^{avail}$ . Then, we let  $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$  be the dataset that consists of all feature-label pairs that exist and  $\mathcal{D}^{avail} = (\mathcal{X}^{avail}, \mathcal{Y}^{avail}) \subset \mathcal{D}$  be the subset that is available to us.

## Embedding networks

Assuming that our dataset  $\mathcal{D}^{avail}$  is representative for all values of interest  $\mathbf{y}_{t,s}$ , i.e. that samples from  $\mathcal{D}^{avail}$  are identically and independently distributed (iid) with some probability distribution  $P(\mathbf{x}_{t,s},\mathbf{y}_{t,s})$ , allows us to learn a functional relationship  $f: \mathcal{X} \to \mathcal{Y}$  using gradient descent algorithms. Our goal is then to generalize as well as possible on data points that are not in  $\mathcal{D}^{avail}$ , i.e. we want to perform well on spatial, temporal and spatio-temporal prediction tasks. Here, we consider modular neural network prediction models. For each individual application and its feature types, a large variety of architectures can be sensible. The architectures that we consider require a multi-input structure with at least one separate input for each of the feature types  $\mathbf{x}_t$ ,  $\mathbf{x}_s$  and  $\mathbf{x}_{st}$ , as far as these are available for the application at hand.

Figure 1, a.) shows a general neural network architecture of this type that we refer to as an *embedding network*. Subsets of in-between layers can be used to shape encoders that embed features into a vector space of arbitrary dimension. The last layer of each encoder is referred to as an embedding layer. Let  $N_{(e)} \in \mathbb{Z}^+$  with  $(e) \in \{t, s, st, x\}$  be the dimension of the vectors into which  $\mathbf{x}_{t,s}$  can be embedded, i.e. the number of nodes of each embedding layer, and  $\mathcal{U}^{(i)}$  be the sets of all embedded vectors in their respective vector spaces with  $(i) \in \{time, space, space-time, joint\}$ . With the proposed network architecture, we can then define the prediction model and its feature encoders as functions

$$f_{NN}: \mathcal{X} \to \mathcal{Y}$$
 (8)

$$enc_{(i)}: \mathcal{X} \to \mathcal{U}^{(i)}$$
 (9)

# The data selection problem

Given  $\mathcal{D}^{avail}$  there usually exists a much larger set of data points that are not available to us. We refer to these as the candidate features  $\mathcal{X}^{cand}$  and their complement labels  $\mathcal{Y}^{cand}$  which together shape the set of candidate data points  $\mathcal{D}^{cand} = (\mathcal{X}^{cand}, \mathcal{Y}^{cand})$ . Our goal is to choose the most informative subset of labels  $\mathcal{Y}^{choice(*)}$  from the large pool of candidate labels  $\mathcal{Y}^{cand}$  such that our overall generalization error decreases the most, without exceeding a given

number of labels, which we refer to as our data budget  $n_{budget}$ . During the data selection process, we assume to have complete access to all existing features  $\mathcal{X}$ , the available labels  $\mathcal{Y}^{avail}$ , but not to any labels from  $\mathcal{Y}^{cand}$ . It hence has to hold that

$$\mathcal{D}^{avail} \cup \mathcal{D}^{cand} = \mathcal{D} \tag{10}$$

$$\mathcal{D}^{avail} \cap \mathcal{D}^{cand} = \varnothing \tag{11}$$

$$\mathcal{Y}^{choice(*)} \subset \mathcal{Y}^{cand}$$
 (12)

The subset of labels that we eventually query without prior information about their values is likely to deviate from the optimal subset  $\mathcal{Y}^{choice(*)}$ ; we refer to the actually queried subset of labels with  $\mathcal{Y}^{choice}$ . The feature-label pairs of queried data points are respectively referred to as  $\mathcal{D}^{choice} = (\mathcal{X}^{choice}, \mathcal{Y}^{choice})$ . One way to query labels is to do so one by one. Another, computationally more effective way to do this, is to use batches of data queries, particularly because we also train our neural network models with batches of data points between each AL iteration. We define the batch size, or number of labels, that are queried in each step of an AL process as  $n_{batch} \in \mathbb{Z}^+$  and the total number of data selection steps as  $n_{iter} \in \mathbb{Z}^+$ . It hence has to hold that

$$n_{iter} \cdot n_{batch} \le n_{budget}$$
 (13)

#### Embedded feature vectors

Given any of the encoders  $enc_{(i)}$  that  $f_{NN}$  incorporates, we can encode each feature vector  $\mathbf{x}_{t,s}$ , and single parts of it  $(\mathbf{x}_t, \mathbf{x}_s, \mathbf{x}_{st})$ , into their embedded vector spaces. Ideally, we expect the distances of these vectors to each other to become increasingly meaningful in the context of our overarching prediction task as we train the actual prediction model  $f_{NN}$  [35, 36, 37, 38, 39]. As our encoders are modules of our prediction model, they are automatically trained each time we apply backpropagation through gradient descent on  $f_{NN}$ . For every feature vector  $\mathbf{x}_{t,s}$  and  $(i) \in \{time, space, space-time, joint\}$ , we can write

$$\forall \mathbf{x}_{t,s} = \begin{pmatrix} \mathbf{x}_t \\ \mathbf{x}_s \\ \mathbf{x}_{st} \end{pmatrix} \in \mathcal{X} \exists \{ \hat{\mathbf{x}}_{(j)} = enc_{(i)}(\mathbf{x}_{(j)}) \}_{(j) = \{(t,s),t,s,st\}}$$
(14)

We refer to predicted labels of data points as

$$\hat{\mathbf{y}}_{t,s} = f_{NN}(\mathbf{x}_{t,s}) \tag{15}$$

## Clusters in embedded feature vectors

Given a set of vectors of the same dimension, we can calculate clusters based on the distances of these vectors to each other using algorithms like K-means or affinity propagation. In order to execute most clustering algorithms, we need to determine the number of desired clusters or a minimum distance of members beforehand. In order to avoid assumptions regarding common distances in the embedded vector spaces, we only consider clustering methods that require a definition of the number of clusters beforehand. We refer to the number of clusters that we set for performing any of these clustering methods with  $n_{clusters}$ . Let  $n_{diverse}^{(i)}$  further be the number of distinct elements in the vector set  $\mathcal{U}^{(i)}$ . For a clustering of embedded vectors to be valid it hence has to hold that

$$n_{clusters} < n_{diverse}^{(i)}$$
 (16)

and for data queries to be sensible furthermore that

$$n_{clusters} << n_{dinerse}^{(i)}$$
 (17)

After clustering the elements of any embedded vector set  $\mathcal{U}^{(i)}$  with  $(i) \in \{time, space, space-time, joint\}$ , we get a first set of vectors  $\mathbf{c}_l^{(i)}$  which describe the center of each cluster with  $l=1...n_{clusters}$ , and a set of values  $m_k^{(i)}$  which describe the membership IDs for each clustered data point with  $k=1...|\mathcal{U}^{(i)}|$ .

#### Distance between features and cluster centers

The distance between any two vectors of the same dimension can be calculated using inner products through e.g. kernel functions or other metrics such as the cosine similarity. Using any of these metrics, we can calculate the distance  $d_j$  of every embedded vector  $\hat{\mathbf{x}}_j$  to its corresponding cluster center  $\mathbf{c}_{m_k^{(i)}}^{(i)}$  with  $(i) \in \{time, space, space-time, joint\}, (j) \in \{t, s, st, (t, s)\}, \text{ and } k = 1, ..., |\mathcal{U}^{(i)}|$  being the element ID that corresponds to the point  $(t, \mathbf{s})$ . Each complete feature vector can be embedded into multiple vector spaces. It further has to hold that

$$0 \le d_{(i)} \le 1 \tag{18}$$

# Embedding entropy

Given the distance of embedded features to their respective cluster centers, we can derive uncertainty measures that we refer to as the *embedding entropy* of each data point. We write the embedding entropy  $H_{(j)}^{NN}$  according to the embedding of the neural network prediction model  $f_{NN}$  for  $\mathbf{x}_{t,s} \in \mathcal{X}^{cand}$  as

$$H_{(j)}^{NN} = d_{(j)}$$
 (19)

where  $(j) \in \{t, s, st, (t, s)\}$ . We name this entropy due to its similarities to existing definitions of thermodynamic system and Shannon entropy. The further

an embedded and clustered feature vector is away from its cluster center, the weaker we assume its membership to this cluster to be. We assume it then to be more likely that a data point belongs to other clusters, hence we are likely closer to a decision boundary and uncertain about its embedding (Figure 1, c.)).

# Batch active learning

Given our metric of informativeness  $H_{(j)}^{NN}$ , we create a pool-based AL method that queries a batch of labels  $\mathbf{y}_{t,s} \in \mathcal{Y}^{cand}$  from the candidate data pool in each iteration. Algorithm 1 shows the pseudo-code for this method.

Starting with a pre-trained prediction model  $f^{NN}$  and all feature encoders that it incorporates (1.), we can choose which feature type and corresponding encoder we want to use for querying candidate data points (2.). Given some data budget  $n_{budget}$  and a maximum number of iterations  $n_{iter}$ , we create a data counter  $c_{budget}$  and an iteration counter  $c_{iter}$  that we set to zero, and leave the set of queried data points empty prior to performing AL (3.). We start our AL iterations by encoding each candidate data point (4.1). If the set of candidate data points is too large for this to be computationally feasible, we can sample a subset of candidate data points uniformly at random. Next, we cluster all embedded feature vectors (4.2) and compute the distances to their respective cluster centers (4.3). We can then calculate the embedding entropy for each candidate point (4.4) and pick the most informative data point from each cluster (4.5). The chosen subset of data points is then used for training our prediction model  $f_{NN}$  (4.6). Next, we remove queried data points from the candidate data pool at a rate  $\delta \in \mathbb{R}$  with  $0 \le \delta \le 1$  (4.7). Here, the rate  $\delta$  is the probability with which we remove a queried data point. A value of  $\delta = 1$ means that all queried data points are removed and a value of  $\delta = 0$  respectively means that all queried data points are kept. Before we continue with the next iteration, we increment our iteration counter by one and our data point counter by the number of newly queried labels among the chosen data points (4.8 - 4.10).

We can highlight two major differences to existing AL methods. First, we remove queried data points from the candidate data set at some rate  $\delta$  only; this allows us to re-use data points so as to implicitly remove any ongoing and initial bias of our model. Second, we set the number of clusters in our features equal to the batch size of data points that we want to query in each iteration; this allows us to implicitly add an instance of importance sampling by building more clusters where data points are densely populated, but also allows us to cope with imbalanced data by choosing batch sizes sufficiently small.

```
1. Train f^{NN} on D^{avail}.

2. Choose one (j) \in \{t, s, st, (t, s)\} and set its corresponding (i) \in \{time, space, space-time, joint\}.

3. c_{budget} = 0; c_{iter} = 0; \mathcal{D}^{choice} = \varnothing.

while c_{budget} < n_{budget} and c_{iter} \leq n_{iter} do
\begin{vmatrix} \mathbf{if} & | \mathcal{X}^{cand} | >> n_{budget} \\ | & | \mathcal{X}^{choice} | >> n_{budget} \\ | & | \mathcal{X}^{choice}
```

Algorithm 1: A pseudo-code of the proposed batch AL method

## Central hypothesis

Let  $error(\hat{\mathbf{y}}_{t,s}, \mathbf{y}_{t,s})$  be the loss function that is used to evaluate our model performance and update model weights through backpropagation on  $f_{NN}$  based on the true labels  $\mathbf{y}_{t,s} \in \mathcal{Y}$  and their complement predictions  $\hat{\mathbf{y}}_{t,s} = f_{NN}(\mathbf{x}_{t,s})$ . We hypothesise that if

- 1. a neural network model  $f_{NN}$  is trained to learn  $f_{NN}: \mathcal{X}^{avail} \to \mathcal{Y}^{avail}$
- 2. any modules of  $f_{NN}$  can be used to create feature encoders  $enc_{(i)}$

then it holds that

$$\exists enc_{(i)} : \sum_{k=1}^{|\mathcal{D}^{cand} \setminus \mathcal{D}^{choice}|} error(f_{NN}^{AL,(i)}(\mathbf{x}_k), \mathbf{y}_k) \le \sum_{k=1}^{|\mathcal{D}^{cand} \setminus \mathcal{D}^{choice}|} error(f_{NN}^{PL}(\mathbf{x}_k), \mathbf{y}_k)$$
(20)

while

$$|\mathcal{Y}_{AL(i)}^{choice}| \le |\mathcal{Y}_{PL}^{choice}| \tag{21}$$

with  $(x_k, y_k) \in \mathcal{D}^{cand} \setminus \mathcal{D}^{choice}$ ,  $(i) \in \{time, space, space-time, joint\}$  and  $f_{NN}^{PL}$  being a copy of  $f_{NN}$  that is trained on data that is randomly chosen from  $\mathcal{D}^{cand}$  representing PL, and  $f_{NN}^{AL(i)}$  being a copy of  $f_{NN}$  that is trained using

Algorithm 1 representing AL. The sets  $\mathcal{Y}^{choice}_{AL(i)}$  and  $\mathcal{Y}^{choice}_{PL}$  respectively refer to the queried labels when applying AL with Algorithm 1 and the PL benchmark.

#### Empirical tests of central hypothesis

We test our central hypothesis empirically. In the supplementary information, we document the executed code for producing the numeric and visual results of our hypothesis test in form of Jupyter notebook sessions saved in .html format. Here, we describe our data and how we split it into training, validation and testing sets, our prediction model, our loss function, and the details of the experiments that we conduct.

#### Feature and label data

We are given the electric consumption measurements of about 100 buildings in Switzerland in 15-min steps from local distribution system operators. Using the geographic coordinates of these buildings, we further collect aerial imagery of each building with a resolution of 25 cm per pixel [51]. We then cluster all buildings that are in a distance of at most 1 km to each other. For each cluster of buildings, we calculate the cluster centers and collect a total of nine meteorological time-series measurements from reanalysis data for each of these clusters with one hour accuracy [52, 53]. The meteorological values that we use consist of air density in kg/m³, cloud cover, precipitation in mm/hour, ground-level solar irradiance in W/m², top of atmosphere solar irradiance W/m², air temperature in °C, snowfall in mm/hour, snow mass in kg/m² and wind speed.

We predict the next 24 hours of electric consumption. For each of the nine meteorological conditions, we consider a historic time window of 24 hours. Time stamps are ordinal encoded, and contain information about the month, day, hour and quarter-hour in which the corresponding electric consumption of a building occurs. Images of buildings are processed using histograms of their pixel values with 100 bins for each image channel (red, green, blue). We can hence set the dimensions of the feature and label vectors to

$$D_y = 4 \cdot 24 = 96 \tag{22}$$

$$D_t = 4 (23)$$

$$D_s = 3 \cdot 100 = 300 \tag{24}$$

$$D_{st} = 9 \cdot 24 = 216 \tag{25}$$

$$D_x = 4 + 300 + 216 = 520 \tag{26}$$

Different than in spatio-temporal point processes, we do not condition our predictions on any historic events about the labels [54]. This means that we

do not use any past values of our labels as features for making predictions. It allows us to make predictions about any point in time and space, without having ground truth data a priori after an initial training of our prediction model.

## Training, validation and testing data

Given a number of data points that are available to us, we create training, validation and testing data for our hypothesis test. The training data is used to fit our prediction model prior to performing AL. The validation data is used to avoid that we overfit our model to the training data through early stopping. The testing data is used as the candidate data pool on which we perform AL to extend our prediction model. We separate our testing data into spatial, temporal and spatio-temporal prediction tests. We use 3% of our data for initial training, 6% for validation, and 91% for testing. We further split our testing data such that 23% of it represent spatial predictions, another 23% temporal predictions and 54% spatio-temporal predictions. In the following, we will refer to training, validation and testing data with  $\mathcal{D}^{train}$ ,  $\mathcal{D}^{val}$  and  $\mathcal{D}^{test}$ . We let  $\mathcal{D}$  be our entire feature-label space and can write for initially available and candidate data that

$$\frac{|\mathcal{D}^{avail}|}{|\mathcal{D}|} = 0.03 \tag{27}$$

$$\frac{|\mathcal{D}^{cand}|}{|\mathcal{D}|} = 0.91\tag{28}$$

## Neural network prediction models and feature encoders

We construct our neural network prediction model from multiple subnetworks (Figure 1, a.)). The modular network which processes our meteorological data consists of an one-dimensional convolutional neural network layer with 16 filters. The modular networks which process the time stamp data and the histograms of building image pixels each contain one densely connected hidden layer with 1,000 nodes. The joint encoder concatenates the outputs of each of these modular networks and adds another densely connected hidden layer with 1,000 nodes. All embedding layers consist of 100 nodes. The entire prediction model then takes the output of the joint encoder and adds another layer of 1,000 densely connected nodes before mapping the joint inputs to the desired output with 96 densely connected nodes. In total, our model contains 10,744,600 trainable and zero non-trainable parameters. For the encoder outputs of all  $(e) \in \{t, s, st, x\}$ , we can write

$$N_{(e)} = 100 (29)$$

## Loss function

We use the root mean squared error (RMSE) between predicted  $(\hat{\mathbf{y}}_{t,s})$  and true labels  $(\mathbf{y}_{t,s})$  as our loss  $error(\hat{\mathbf{y}}_{t,s},\mathbf{y}_{t,s})$ . In each epoch of training and validation,

as well as for each test, we calculate the total loss function  $loss(\mathcal{D}^{(d)})$  as the average loss of all data points in the respective data sets  $D^{(d)}$ , where  $(d) \in \{train, val, test\}$ . With  $j = 1, ..., |\mathcal{D}^{(d)}|$  being the element ID that corresponds to the point  $(t, \mathbf{s})$  in  $\mathcal{D}^{(d)}$ , we can write for all pairs of  $(\hat{\mathbf{y}}_j, \mathbf{y}_j) \in \mathcal{D}^{(d)}$  that

$$error(\hat{\mathbf{y}}_{t,s}, \mathbf{y}_{t,s}) = \sqrt{\frac{\sum_{k=1}^{D_y} (y_{t,s,k} - \hat{y}_{t,s,k})^2}{D_y}}$$
 (30)

and

$$loss(\mathcal{D}^{(d)}) = \frac{\sum_{j=1}^{|\mathcal{D}^{(d)}|} error(\hat{\mathbf{y}}_j, \mathbf{y}_j)}{|\mathcal{D}^{(d)}|}$$
(31)

## Experiments

We assume our data budget to be 50% of the size of our candidate data pool, i.e. we want to choose the more informative half of candidate data points. We perform ten iterations of the above Algorithm 1 where we query 10% of our data budget in each iteration. We train our prediction model for 30 epochs and use an early stopping patience of 10 epochs when training our prediction model on the initially available data and in each iteration of Algorithm 1. We can write

$$n_{budget} = 0.5 \cdot |\mathcal{D}^{cand}| \tag{32}$$

$$n_{iter} = 10 (33)$$

$$n_{clusters} = n_{batch} = 0.1 \cdot n_{budget} = 0.05 \cdot |\mathcal{D}^{cand}|$$
 (34)

We use the K-means++ algorithm to cluster embedded feature vectors, and the Gaussian kernel to calculate the distance between each embedded feature vector and its cluster center. Using the Gaussian kernel is equivalent to using the Euclidean distance for the performance of Algorithm 1 as both lead to the same hierarchy of vector distances. Given the embedded vector set  $\mathcal{U}^{(i)}$  with  $(i) \in \{time, space, space-time, joint\}$ , the vectors that describe the center of each cluster  $\mathbf{c}_l^{(i)}$  with  $l=1...n_{clusters}$  and a membership ID  $m_k^{(i)}$  for each embedded feature  $k=1...|\mathcal{U}^{(i)}|$ , we calculate the embedding entropy  $H_{(j)}^{NN}$  for every feature type  $(j) \in \{t, s, st, (t, s)\}$  and point in time-space  $(t, \mathbf{s})$  as

$$H_{(j)}^{NN} = d_{(j)} = exp\left(-\frac{||\hat{\mathbf{x}}_{(j)} - \mathbf{c}_{m_k^{(i)}}^{(i)}||^2}{N_{(i)}}\right)$$
(35)

We test Algorithm 1 for every partial feature vector, and the entire feature vector separately. Since our labels have a similar dimension  $(D_y = 96)$  as our embedded features  $(N_{(e)} = 100)$ , we use the predicted labels  $(\hat{\mathbf{y}}_{t,s})$  as our jointly embedded feature vectors  $(\hat{\mathbf{x}}_{t,s})$ . Alternatively, one can also design the joint

feature encoder to contain all layers of the entire prediction network except for the output layer as proposed in [49] to achieve similar results. We also evaluate a scenario in which we query candidate data points based on the embedding entropy of their true labels  $\mathbf{y}_{t,s}$ . We conduct our tests for the two cases that we mainly distinguish: first, we remove each queried point from the candidate data pool at the end of each AL iteration ( $\delta = 1$ ); second, we keep queried data points in the candidate pool throughout all AL iterations ( $\delta = 0$ ).

The experiments that we conduct are extensively documented and provided in four supplementary files. These files are named with 'HypothesisTest1.html' to 'HypothesisTest4.html'. The first two files contain the experiments that we conduct for  $\delta=1$  and  $\delta=0$  when we validate our average RMSE against unqueried candidate data points. The remaining two files respectively contain the experiments for  $\delta=1$  and  $\delta=0$  when we validate our average RMSE against queried and unqueried candidate data points. Furthermore, we make our result figures and tables reproducible through an additional supplementary file that is named 'ResultFigures.html'.

# Data availability

The electric consumption data and aerial imagery of buildings that we use in our empirical tests are privacy breaching and are therefore not made available. Meteorological data can be downloaded from the renewables.ninja website [55].

# Code availability

We provide step-by-step instructions for implementing the algorithm that we propose and the prediction models that it contains as a Jupyter notebook session under a MIT license on Github [56].

# Acknowledgements

We thank Mohammad Reza Karimi for passionately reflecting with us on the important implications of our findings. We further acknowledge funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 837089, for the SENTINEL project.

# Author contributions

Arsam Aryandoust conceptualised the research, designed and implemented all methods, conducted all analyses, and drafted the manuscript. Stefan Pfenninger edited and revised the manuscript and reviewed the code.

# Competing financial interests

The authors declare no competing financial interests.

# References

- [1] Rolnick, D. & et al. Tackling Climate Change with Machine Learning. Preprint at https://arxiv.org/abs/1906.05433 (2019).
- [2] Settles, B. Active Learning Literature Survey. Computer Sciences Technical Report 1648, http://digital.library.wisc.edu/1793/60660 (2010).
- [3] Dasgupta, S. Two faces of active learning. *Theoretical Computer Science* **412**, 1767–1781 https://doi.org/10.1016/j.tcs.2010.12.054 (2011).
- [4] Kumar, P. & Gupta, A. Active Learning Query Strategies for Classification, Regression and Clustering: A Survey. *Journal of Computer Science and Technology* **35**, 913-945 https://doi.org/10.1007/s11390-020-9487-4 (2020).
- [5] Lewis, D. D. & Catlett, J. Heterogeneous Uncertainty Sampling for Supervised Learning. *Machine Learning Proceedings* 11, 148-156 https://doi.org/10.1016/B978-1-55860-335-6.50026-X (1994).
- [6] Fujii, T., Inui, K., Tokunaga, T. & Tanaka, H. Selective sampling for example-based word sense disambiguation. *Computational Linguistics* 24 (4), 573-597 (1998).
- [7] Scheffer, T., Decomain, C. & Wrobel, S. Active Hidden Markov models for Information Extraction. *IDA* , 309–318 (2001).
- [8] Hwa, R. Sample Selection for Statistical Parsing. Computational Linguistics 30 (3), 253–276 https://doi.org/10.1162/0891201041850894 (2004).
- [9] Lindenbaum, M., Markovitch, S. & Rusakov, D. Selective sampling for nearest neighbor classifiers. *Machine Learning* **54**, 125-152 (2004).
- [10] Culotta, A. & McCallum, A. Reducing labeling effort for structured prediction tasks. AAAI 2, 746-751 (2005).
- [11] Körner, C. & Wrobel, S. Multi-class ensemble-based active learning. *ECML*, 687-694 (2006).
- [12] Schein, A. I. & Ungar, L. H. Active learning for logistic regression: An evaluation. *Machine Learning* 68 (3), 235-265 https://doi.org/10.1007/s10994-007-5019-5 (2007).
- [13] Seung, H. S., Opper, M. & Sompolinsky, H. Query by committee. *Proceedings of the ACM Workshop on Computational Learning Theory*, 287-294 https://doi.org/10.1145/130385.130417 (1992).

- [14] Dagan, I. & Engelson, S. Committee-based sampling for training probabilistic classifiers. ICML, 150-157 https://doi.org/10.1016/B978-1-55860-377-6.50027-X (1995).
- [15] Abe, N. & Mamitsuka, H. Query learning strategies using boosting and bagging. *ICML*, 1-9 (1998).
- [16] McCallum, A. & Nigam, K. Employing EM in pool-based active learning for text classification. ICML, 359-367 (1998).
- [17] Muslea, I., Minton, S. & Knoblock, C. A. Selective sampling with redundant views. AAAI, 621-626 (2000).
- [18] Melville, P. & Mooney, R. J. Diverse ensembles for active learning. *ICML*, 584-591 https://doi.org/10.1145/1015330.1015385 (2004).
- [19] Hanneke, S. Theory of Disagreement-Based Active Learning. Foundations and Trends in Machine Learning 7, 31–309 https://doi.org/10.1561/2200000037 (2014).
- [20] MacKay, D. J. C. Information-based objective functions for active data selection. *Neural Computation* 4, 590-604 http://dx.doi.org/10.1162/neco.1992.4.4.590 (1992).
- [21] Cohn, D. A. Neural network exploration using optimal experiment design. *NIPS*, 679-686 (1994).
- [22] Paass, G. & Kindermann, J. Bayesian query construction for neural network models. *NIPS*, 443-450 (1995).
- [23] Cohn, D. A., Ghahramani, Z. & Jordan, M. I. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 129-145 (1996).
- [24] Zhang, T. & Oles, F. J. A probability analysis on the value of unlabeled data for classification problems. *ICML*, 1191-1198 (2000).
- [25] Hoi, S. C. H., Jin, R. & Lyu, M. R. Large-scale text categorization by batch mode active learning. WWW, 633-642 https://doi.org/10.1145/1135777.1135870 (2006).
- [26] Roy, N. & McCallum, A. Toward optimal active learning through sampling estimation of error reduction. *ICML*, 441-448 (2001).
- [27] Zhu, X., & Lafferty, J. Ghahramani, Z. Combining Active Learning and Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. Proceedings of the ICML Workshop on Continuum from Labeled to Unlabeled Data, 58-65 (2003).
- [28] Guo, Y. & Greiner, R. Optimistic active learning using mutual information. *IJCAI*, 823-829 (2007).

- [29] Moskovitch, R. & et al. Improving the detection of unknown computer worms activity using active learning. KI (2007).
- [30] Settles, B., Craven, M. & Ray, S. Multiple-instance active learning. NIPS, 1289–1296 (2007).
- [31] Settles, B. & Craven, M. An analysis of active learning strategies for sequence labeling tasks. EMNLP, 1070–1079 https://doi.org/10.3115/1613715.1613855 (2008).
- [32] Nguyen, H. T. & Smeulders, A. Active learning using pre-clustering. *ICML* https://doi.org/10.1145/1015330.1015349 (2004).
- [33] Xu, Z., Akella, R. & Zhang, Y. Incorporating diversity and density in active learning for relevance feedback. *ECIR*, 246–257 (2007).
- [34] Dasgupta, S. & Hsu, D. J. Hierarchical sampling for active learning. *ICML* https://doi.org/10.1145/1390156.1390183 (2008).
- [35] Mikolov, T., Chen, K., Corrado, G. & Dean, J. Efficient Estimation of Word Representations in Vector Space. Preprint at https://arxiv.org/abs/1301.3781 (2013).
- [36] Frome, A. & et al. DeViSE: A Deep Visual-Semantic Embedding Model. NIPS, 2121–2129 (2013).
- [37] Pennington, J., Socher, R. & Manning, C. D. GloVe: Global Vectors for Word Representation. EMNLP https://doi.org/10.3115/v1/D14-1162 (2014).
- [38] Perozzi, B., Al-Rfou, R. & Skiena, S. DeepWalk: Online Learning of Social Representations. Preprint at https://arxiv.org/abs/1403.6652 https://doi.org/10.1145/2623330.2623732 (2014).
- [39] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Preprint at https://arxiv.org/abs/1810.04805 (2019).
- [40] Ghazi, B., Panigrahy, R. & Wang, J. R. Recursive Sketches for Modular Deep Learning. *ICML* (2019).
- [41] Geifman, Y. & El-Yaniv, R., Deep active learning over the long tail. Preprint at https://arxiv.org/abs/1711.00941 (2017).
- [42] Wang, M., Fan, M., Zhang, Z.-H. & Wu, Y.-X. Active learning through density clustering. *Expert Systems with Applications* **85**, 305-317 https://doi.org/10.1016/j.eswa.2017.05.046 (2017).
- [43] Siddiqui, Y., Valentin, J. & Nießner, M. ViewAL: Active Learning with Viewpoint Entropy for Semantic Segmentation. *CVPR* (2020).

- [44] Beygelzimer, A., Dasgupta, S. & Langford, J. Importance Weighted Active Learning. *ICML*, 49–56 (2009).
- [45] Huijser, M. W. & Van Gemert, J. C. Active Decision Boundary Annotation with Deep Generative Models. *CVPR* (2017).
- [46] Sener, O. & Savarese, S. Active Learning for Convolutional Neural Networks: A Core-Set Approach. *ICLR* (2018).
- [47] Smith, K. On the standard deviations of adjusted and interpolated values of an observed polynomial function and it's constant and the guidance they give towards a proper choice of the distribution of observations. *Biometrika* 12, 1-85 (1918).
- [48] Kiefer, J. & Wolfowitz, J. Optimum designs in regression problems. *Annals of Mathematical Statistics* **30**, 271–294 (1959).
- [49] Ash, J. T., Zhang, C., Krishnamurthy, A., Langford, J. & Agarwal, A. Deep Batch Active Learning by Diverse, Uncertain Gradient Lower Bounds. ICLR (2020).
- [50] Karimi, M. R., Gürel, M., N., Karlaš, B., Rausch, J., Zhang, C. & Krause, A. Online Active Model Selection for Pre-trained Classifiers. Preprint at https://arxiv.org/abs/2010.09818 (2020).
- [51] Federal Office of Topography GeoVITE User-friendly Geodata Service. https://geovite.ethz.ch/, (2020).
- [52] Pfenninger, S. & Staffel, I. Long-term patterns of European PV output using 30 years of validated hourly reanalysis and satellite data. *Energy* **114**, 1251-1265 (2016).
- [53] Staffel, I. & Pfenninger, S. Using Bias-Corected Reanalysis to Simulate Current and Future Wind Power Output. *Energy* **114**, 1224-1239 (2016).
- [54] Gonzalez, J. A., Rodriguez-Cortes, F. J., Cronie, O. & Mateu, J. Spatiotemporal point process statistics: A review. Spatial Statistics 18 (B), 505-544 (2016).
- [55] Pfenninger, S. & Staffel, I. https://www.renewables.ninja/, (2020).
- [56] Aryandoust, A. Active machine learning for spatio-temporal predictions using feature embedding. https://github.com/ArsamAryandoust/ActiveLearning, (2020).